

pthread\_mutex\_lock( pthread\_mutex\_t \* mutex )

pthread\_mutex\_unlock( pthread\_mutex\_t \* mutex )

Eks:

pthread\_mutex\_t lock; ~~PTHREAD\_MUTEX\_INITIALIZER~~

pthread\_mutex\_lock(&lock)

x = x + 1;

pthread\_mutex\_unlock(&lock)

HUSK! Deklarer locks ~~locally~~,  
globalt

Hva er pthread\_t strukturen?

Lock deklarereres

pthread\_mutex\_t lock;

## 7.3 Locks

### Design mål

- Mutual exclusion
- Rettferdighet
- Ytelse.

Interrupts kan være et problem, så hvorfor ikke skru dem av?  
Da mister OS-en kontroll, og en app kan ta over hele PC-en.

### Bruk av flags?

typ:

lock → flagg = 1

Kan ikke gjennomføres atomisk.

### Test and set

Er gjort atomisk med hjelp av HW.

X86: xchq

### Compare and swap

Er gjort atomisk med hjelp av HW.

X86: cmpxchq (needs lock prefix)

Trenger ikke lock prefir med len 1CPU

### Spin or switch

- Spinlocks er dårlig for performance, da CPU-en gjør ingenting.
- Kan yielde CPU.
- Spin kan funke om spin-time er kort.
- Kan kombineres til spin then yield.

## 7.4 Deadlock

RIP... alt stoppet:(

Lösning med to enkle regler

1. Nummerer ressursene.
2. Alle spør om ressursene i samme rekkefølge.

← Dining philosophers  
-problem

# 8. Condition Variables and Semaphores

## 8.1 Condition Variables

Vent på en kondisjon som en annen tråd vil trigge.

### Producer Consumer

Eksampl:

- Multithread webserver.
- Linux Command pipeline.
- Network Interface traffic.
- Message queue based applications.

### Producer Consumer Regler

- Bare en tråd kan tildeles bufferen av gangen.
- Producer kan ikke fylle på når det er fullt.
- Consumer kan ikke ta noe når det er tørt.

**BRUK ALLTID while IKKE if FORÅSJEKKE CONDITIONS  
LES 33.2**

Signal til alle threads?

- `pthread_cond_broadcast()`

## 8.2 Semaphore

En semaphore har en int-verdi som kan atomisk økes og minskes.

### Regler

- Teller opp og ned "automatisk" når funksjonene brukes
- Teller man ned når en semaphore er 0 eller under, så blokkeres den (venter)
- Teller man opp på en semaphore som blir ~~over~~ 0 eller over så blir en prosess/tråd unblocked.
- Det negative tallet representerer antall ventende tråder.

Linux bruker ikke negative verdier i semaphorer.

### Binær semaphore

- Kan brukes som mutex lock

### Ordering

- Bruk semaphores for å bestemme rekkefølgen på ting

### Producers-Consumer

Ganske greit for å passe på buffer.

full og empty semaphores.

Før mutual exclusion kan man sette en binær semaphore inn.

### Reader-Writer

Legg inn et tegn til at writer ventet.

## 8.3 Barrier

- Nyttig for å vente på et sett tråder
- `pthread_barrier_wait()`

## 8.4 Monitor

Noen språk gjør synkronisering mye lettere

- Java keyword `synchronized`
- Compiler tilsker det for oss

## 8.5 Deadlock

Gjelder samme som i kap. 7

## Semaphore

- sem\_wait

- teller ned

- teller til under 0 → blokkert

- sem\_post

- teller opp

- teller til over 0 → frigjør en bøn

En semafor som starter på 0 er en løs, men uten eier, så kan låse opp for hvem som helst.

## LØS Reader-Writer

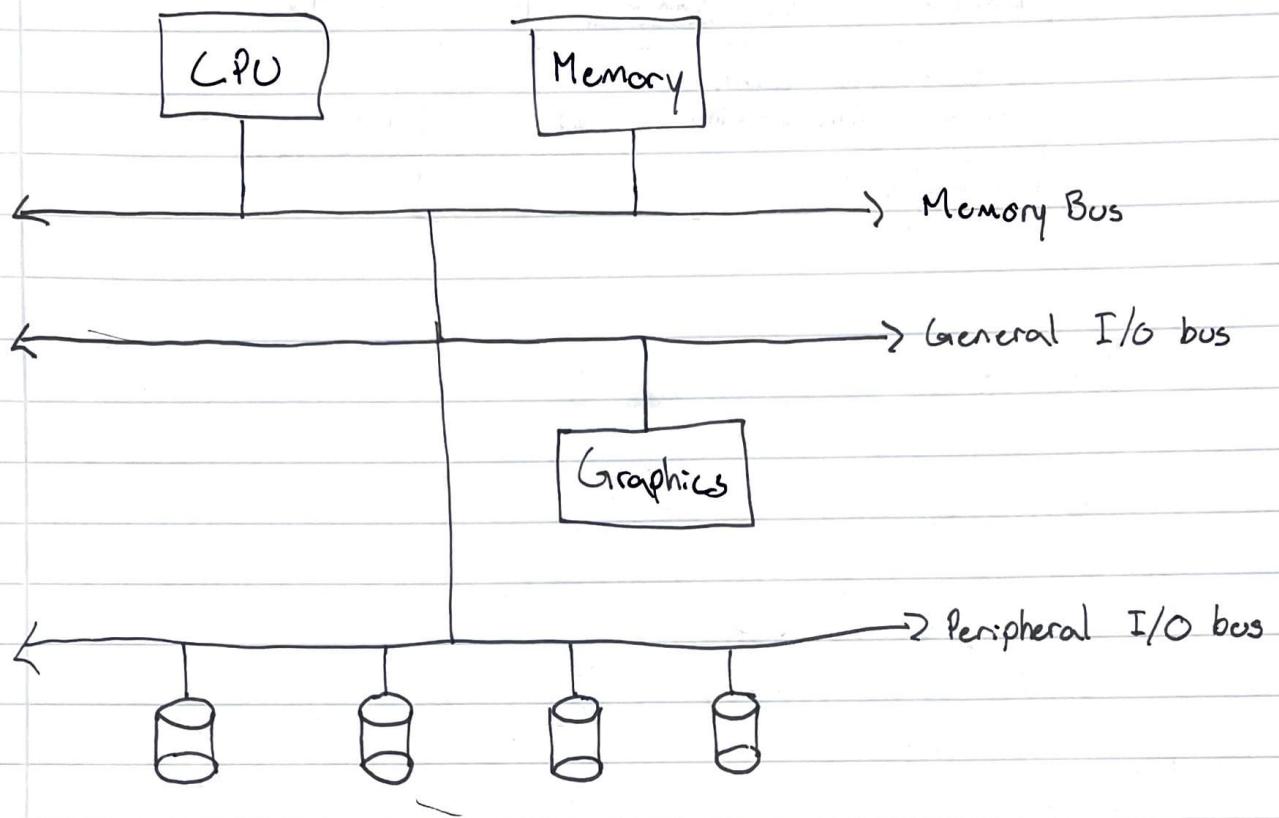
navn  
↓

sem\_t mutex;

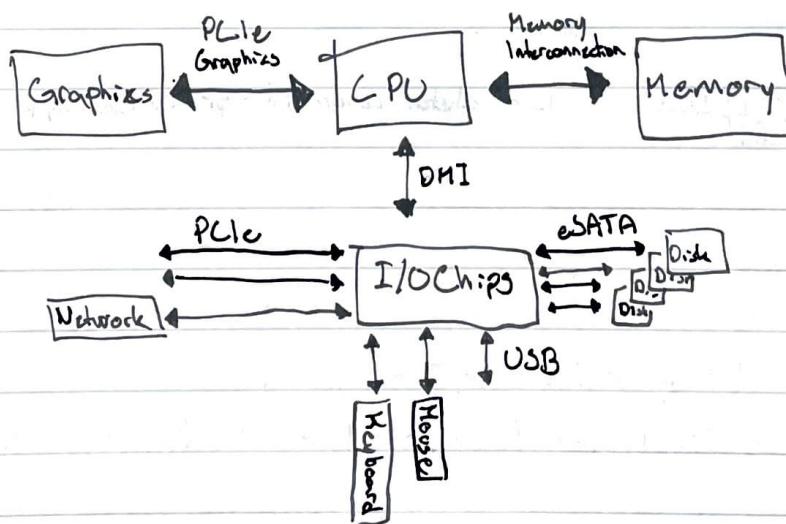
verdi:  
↓

sem\_init(&mutex, 0, 1)

# I/O and RAID



## Moderne



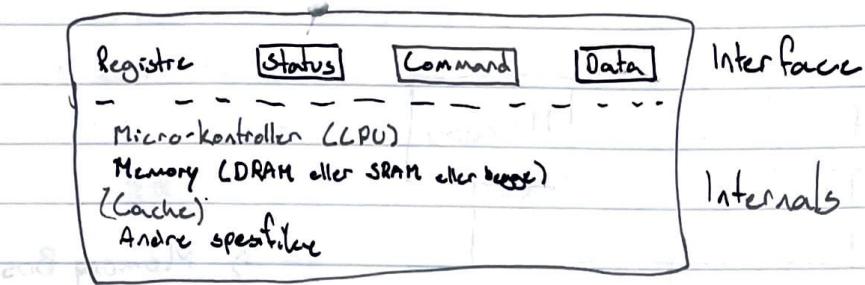
DMI - Direct Media Interface

eSATA - Storage Interface (up to 600 GB/s)

USB - Universal Serial Bus (up to 5 GB/s)

PCIe - High performance devices (up to 128 GB/s)

## I/O enheter



Instruksjoner:

while (STATUS == Busy); (vent)

write (DATA to Register);

write (COMMAND to Register);

Nøkkeljøres kommandoen

while (STATUS == Busy);

Tre måter å gjøre I/O:

Programmed I/O (spin wait)

Interrupt-based I/O



Lar enheten sende interrupt når klar for/ferdig I/O

Direct Memory Access (DMA)

CPU på start og sluttet av I/O-oppgaver.

I/O outsources til DMA-kontroller. Frakutter data fra minne til I/O

## Hvordan kontakter I/O

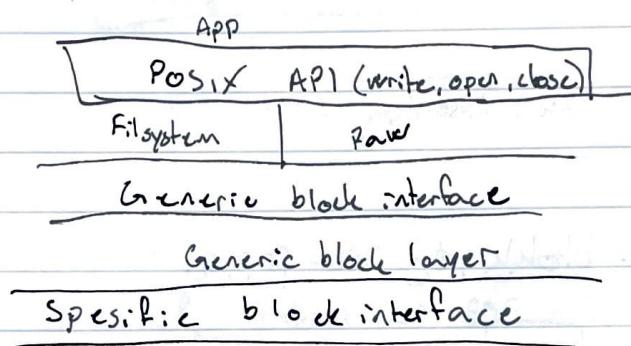
### I/O instructions

Instruksjoner "in" og "out". Med adressrom basert på porter (tilsvarende UDP/TCP) 0 - 65 535

### Memory mapped I/O

- Ligger i fysiske minne
- Kan skrive til I/O likt som man skriver til RAM

NB! Memory mapped file vs. Memory mapped I/O



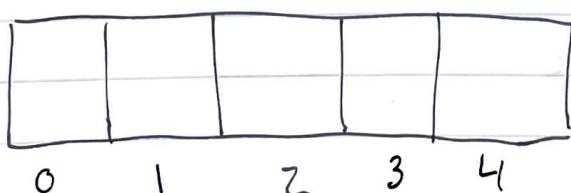
### Blokker og sector og page

Blokk-addresse gir til en blokk (ofte 4KB)

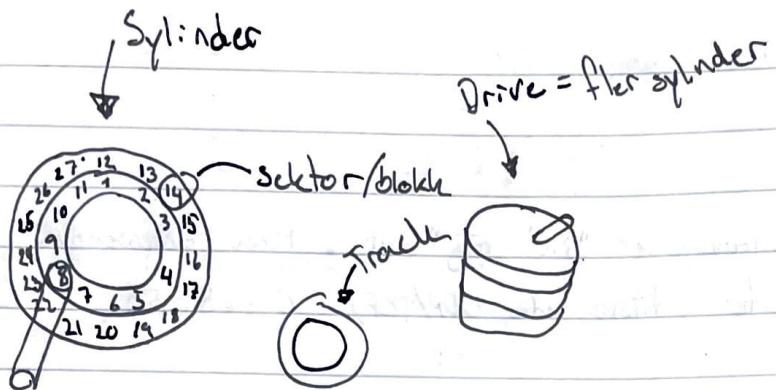
Blokker caches i RAM, ikke filer, ikke bit.

Virtuell maskin har harddisk i en fil som har en blokk-device.

Disk:



## Hard Disk



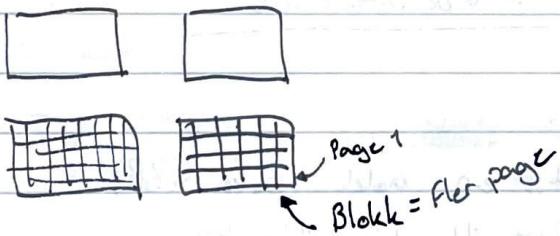
Kan kun skrive til en blokk av gangen (ikke mindre)  
typisk 512B eller 4kB

- Seek er å bruke spor
- 

$$T_{IO} = T_{seek} + T_{rotation} + T_{transfer}$$

## SSD

Kan bare slette blokk, og ikke page



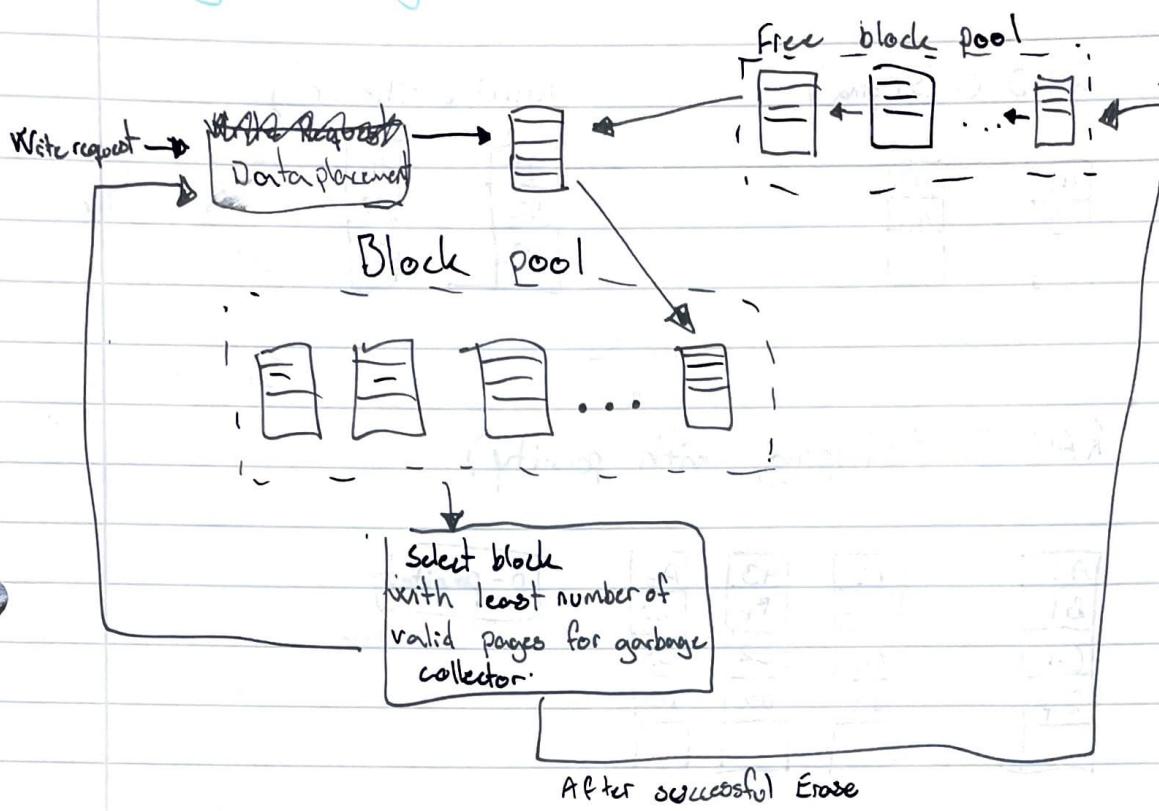
SLC - Single-level cell

MLC - Multi-level cell

TLC - Triple-level cell

Deslites ut over bruk

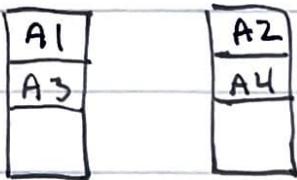
## Unngå slitasje



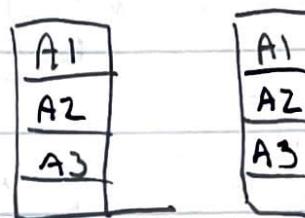
TRIM kommando - fjerner/sletter den dataen som er slettet  
fra bakgrunnen. SSD kan ikke overskrive, så  
må slettes først.

# RAID

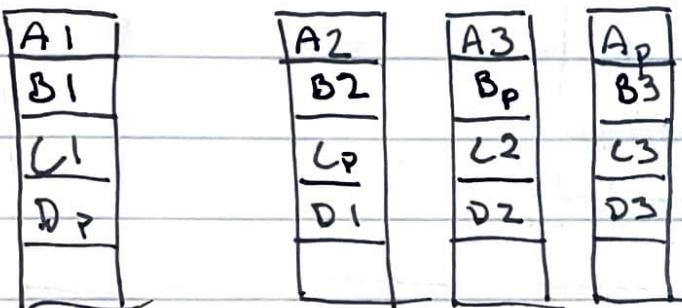
RAID 0 (Striping)



RAID 1 (Mirror)



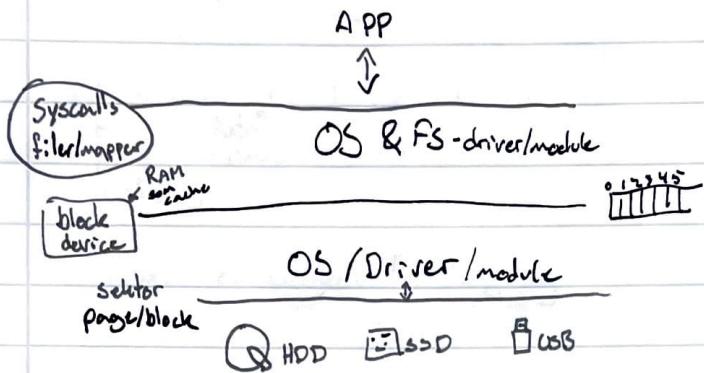
RAID 5 (Striping with parity)



p-paritet

NB! Bruk RAID 6

# Filsystemer



## API

- open(), openat(), creat(), returnerer en **file descriptor**
- read(), **lese** fra filbeschreveren til bufferet og returner
- write()
- close()

## File descriptor

- stdin, stdout, stderr (0, 1, 2)
- Videre fra Ø og opp
- Pipe om dirigerer stdin og stdout

## Sync

- Skrives ikke alltid rett til permanent minne
- fsync() forcer

## Metadata

fil = data + metadata

inode = metadata

## Directories

- `mkdir()`
- `opendir()`
- `readdir()`
- `closedir()`
- `rmdir()`
- " .. " og ". " items

## Links

- `hardlink (link()) (ln)`
  - samme inode
- `symbolic link symlink() (ln -s)`
  - peker til en annen fil
  - er en egen fil
  - egen inode

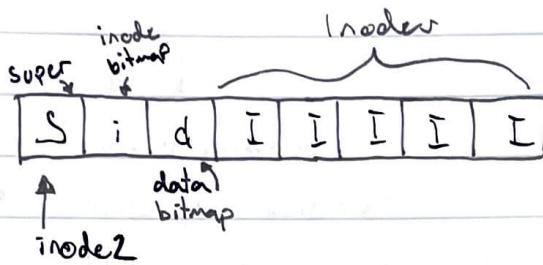
## Permission bits

- `rwx rwx rwx`
- `chmod()`, `chown()`
- SetUID (run som root)
  - SetGID
  - Sticky bit

## Mount

- `mkfs`
- man 2 `mount()`/`umount()`

# I bruk



Resten er blokkene for lagring

## Finne innhold i fil

### 1. Finn inode

Anta 32-bit blokkaddr  
(4B)

60-byte til 15 blokk addr

De første 12 peker til blokkene

$$12 \cdot 4KB = 48KB$$

Hva om du vil ha mer?

## Indirekte adressering

- Single indirect
- Double indirect
- Trippel indirect

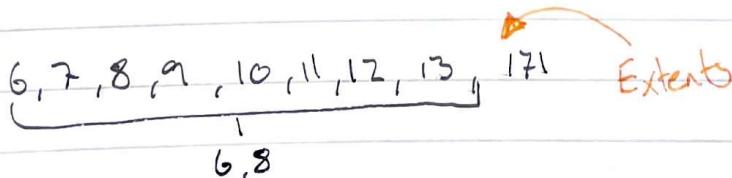
Nytta eks: 32-bit blokk addr 8KB blokkestr

$$\frac{8KB}{4B} = \frac{2^{13}}{2^2} = 2^{11}$$

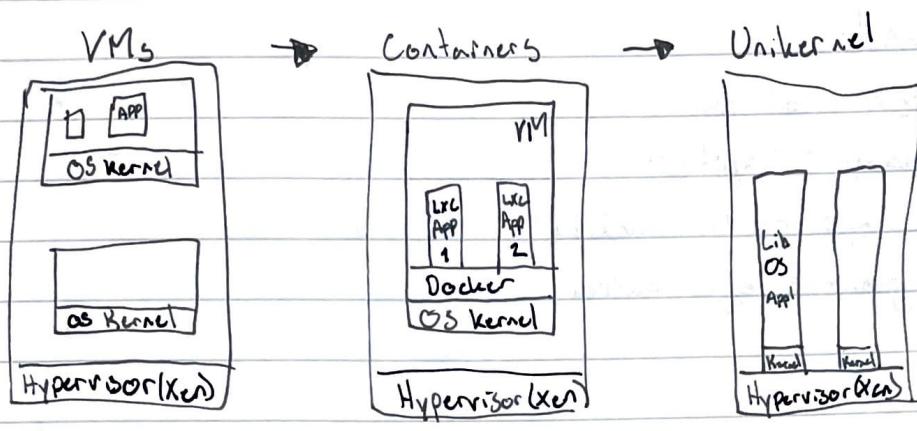
$$\text{single} \rightarrow 2^{11} \cdot 2^{13}B = 2^{24}B = 16MB$$

$$\text{double} \rightarrow 2^{11} \cdot 2^{11} \cdot 2^{13}B = 2^{35}B = 32GB$$

$$\text{trippel} \rightarrow (2^{11})^3 \cdot 2^{13} = 2^{46} = 64TB$$



# Virtuelle Masiner og Konteiner



Hvor

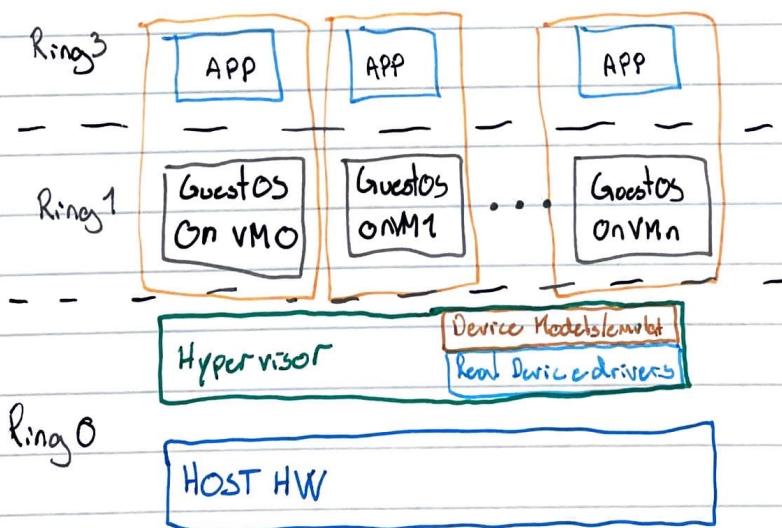
Sensitive Instruction - kan bare utføres i kernel mode

Priviligert Instruksjon - vil trappe dersom den utføres utenfor kernel.

"A machine is virtuizable only if the sensitive instructions are a subset of the sensitive instructions."  
privileged

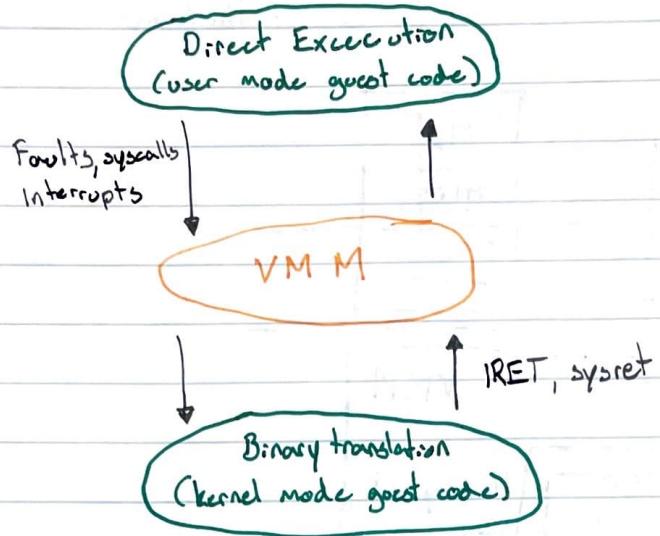
## Hypervisors

### Hypervisor Arkitektur

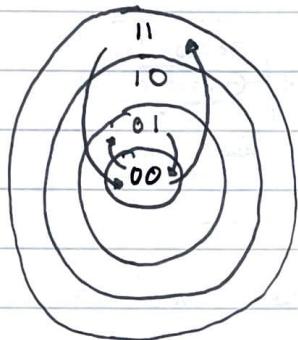


## Binar oversetting

Brukes på x86, fordi quota under hra... ikke stemmer



→ Binary Translation skjer når guest OS-et brukes.



Translator Cache hjelper med ytelse.

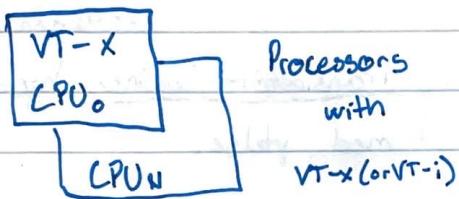
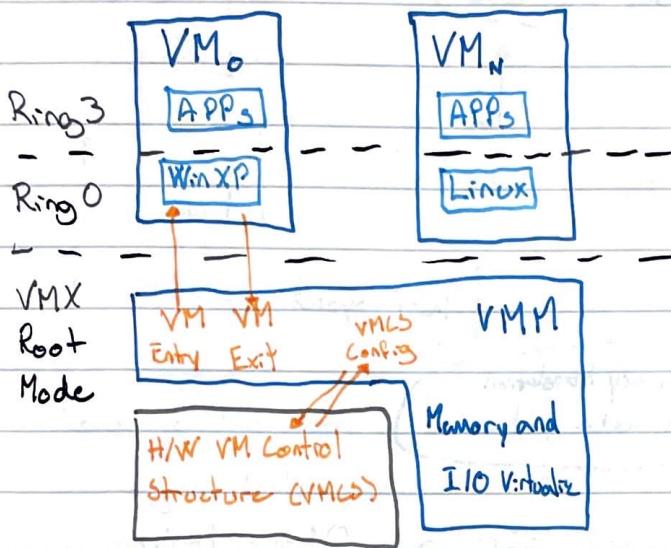
## Xen hypervisor

Paravirtualisering.

Helt OS-et er allerede oversatt.

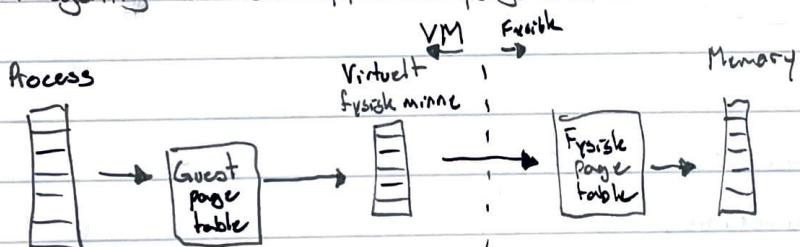
## Hardware virtualisering

Man lar til et nivå (-1). Guest OS fikk gå i kernel mode.



## Memory

Blir egentlig dobbet opp med page tabler

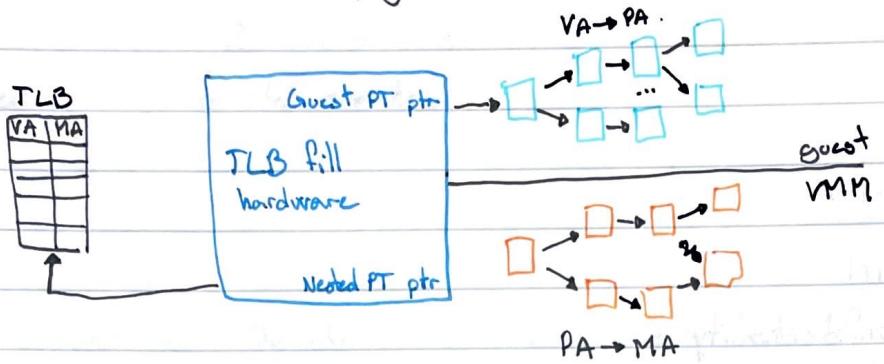


## Shadow page table

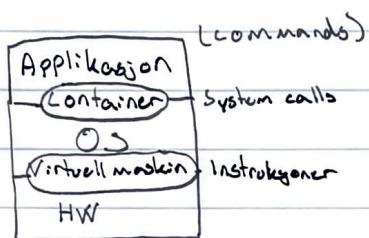
## Nested page table (Hardware)

En SUPER-TLB, som cacher både Guest OS og VMM.

TLB merkes med hvilket page table de tilhører.



## Kontainere



En kontainer er en samling (offte bare en) av prosesser. De er beskyttet med 3 ting:

### C-groups

- Begrenser bruke av ressurser (CPU, minne, I/O...)
- Støttes av OS

## Namepaces

- PIDs, net, mount, ipc
- Støttes av OS

## CoW (Copy on Write)

Copy-on-Write filesystem

## 12. Operating system Security

### 12.1 Intro

- OS-et må være sikert.
  - Om OS-et er usikert så blir alt som kjører på det også usikert.
- OS-et påtvinger sikkerhets-policies (access control).
- OS-et burde hindre begrense skade fra uønsket software.
- OS-et er vanskelig å sikre da det er et stort og komplekt program.

### Mål

- Confidentiality
- Integrity
- Availability

Security policies er regler som sier hva som er, og ikke er lov.

### Design prinsipper

1. Economy of mechanism
2. Fail-safe defaults
3. Complete mediation
4. Open design
5. Separation of privilege
6. Least privilege
7. Least common mechanism
8. Acceptability

## 12.2 Access Control

### Reference Monitor

Hver gang du vil ha tilgang til noe, (åpne en fil), så må denne requesten autoriseres. Autorisasjon må være effektivt med lite overhead og må være riktig.

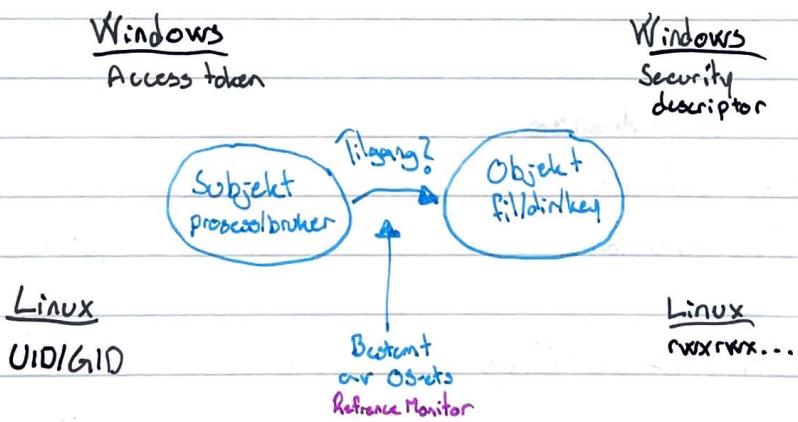
### Capability vs. Access Control List

Capability list er en liste objekter som lagres med subjektet (en bruker eller en prosess), og med tilganger (rwx).

Access Control List er en liste brukere/grupper som lagres hos objektet, også med tilganger (rwx).

ACL er relativt vanlig å bruke.

### Windows og Linux



## MAC/DAC

Discretionary Access Control (DAC) er når brukeren kan bestemme tilgangskontroll. Som med chmod på Linux eller Set-ACL på Windows.

Mandatory Access Control (MAC) er når operativsystemet påtvinger regler som overgår DAC.

## MAC

- Prossesser har et integritetsnivå i sin Access Token.
- Objekter har et integritetsnivå i SACL i Security Descriptoren.
- Reference Monitoren sjekker først integritetsnivå i SACL.
- Prossesser kan ikke lese objekter med høyere integritetsnivå.

En prosess blir startet med et integritetsnivå, og det kan ikke endres mens prosessen kjøres.

SID - hvert integritetsnivå har en egen SID

Default ACL - tilsvarende umask. Setter default permissions.

User SID - eieren av prosessen.

Group SID - grupper prosessen tilhører.

Privileges - Spesielle tilganger knyttet til en bruker. Kan brukes for å gi kort deler av tilgangene til administrator.

System Access Control List

## NTFS

NTFS har flere mulige permissions

- Full Control
- Modify
- Read and execute
- List folder contents
- Read
- Write

Gjelder bare filer. I Windows er ikke alt filer, som på Linux.

## Windows Operation

1. CTRL-ALT-DEL → initierer winlogon
2. Win logon bruker lsass for å autentisere
3. GUI shell explorer med en access token

## User Account Control (UAC)

Mange gamle opper antar de kjører som admin.

UAC gjør:

- Alle admin brukere kjøres med standard bruker privilegier.
  - Medlemsskap; admin er markert DENR.
- FS obj register namespace virtualisering brukes for legacy-software.

Kan få mer tilgang med:

- Run as administrator
- request Execution Level

## Linux operation

1. Login sjekker brukernavn / passord og grupper
  - etc/passwd, etc/shadow
  - etc/groups
2. Starter shell med brukerens UID, GID
3. "sudo" nimer om UAC

## 12.3 Memory Protection

### Buffer overflow

Overskrive en buffer for å skrive andre steder i minne.  
Kan få en prosess til å kjøre farlig kode.

### Forsvar: Stack Canary

En tilfeldig canary-verdi: lagres etter return addressen.  
Dersom den canary-verien er overskrevet, så vet man noe er galt.

### Data Execution Prevention (DEP)

"w XOR x"

### NX-bit

Markerer områder i minne som non-executable. F.eks så man ikke kan skrive kode direkte til stacken.

## Return to libc

I stedet for å ha eksekverbar kode på stacken, kan man gi en return-addresse til en funksjon som ligger et annet sted.

## Defence: Address Space Layout Randomization

Gir tilfeldige addresser til funksjoner og data hver gang et program kjører.